CSE 250
Data Structures

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Lec 27: Tree Rotations

Warm-Up Questions

1. What is the maximum depth of a BST?
2. What is the maximum height of a BST?
3. What is the deepest a BST could be?
4. What is the largest number of edges from root to leaf in a BST?
5. Max depth of a BST????
6. What is the worst case runtime of find, insert, remove on a BST?

Warm-Up Questions

1. What is the maximum depth of a BST? O(n)
2. What is the maximum height of a BST? O(n)
3. What is the deepest a BST could be? O(n)
4. What is the largest number of edges from root to leaf in a BST? O(n)
5. Max depth of a BST???? O(n)
6. What is the worst case runtime of find, insert, remove on a BST? O(n)

Warm-Up Questions

1. What is the maximum depth of a BST? O(n)
2. What is the maximum height of a BST? O(n)
3. What is the deepest a BST could be? O(n)
4. What is the largest number of edges from root to leaf in a BST? O(n)
5. Max depth of a BST???? O(n)
6. What is the worst case runtime of find, insert, remove on a BST? O(n)

O(n) O(n) O(n) O(n) O(n) O(n)

Announcements

- WA4 due Sunday (very useful for midterm)
- Midterm review session held by SAs this Saturday @ 11AM

BST Operations

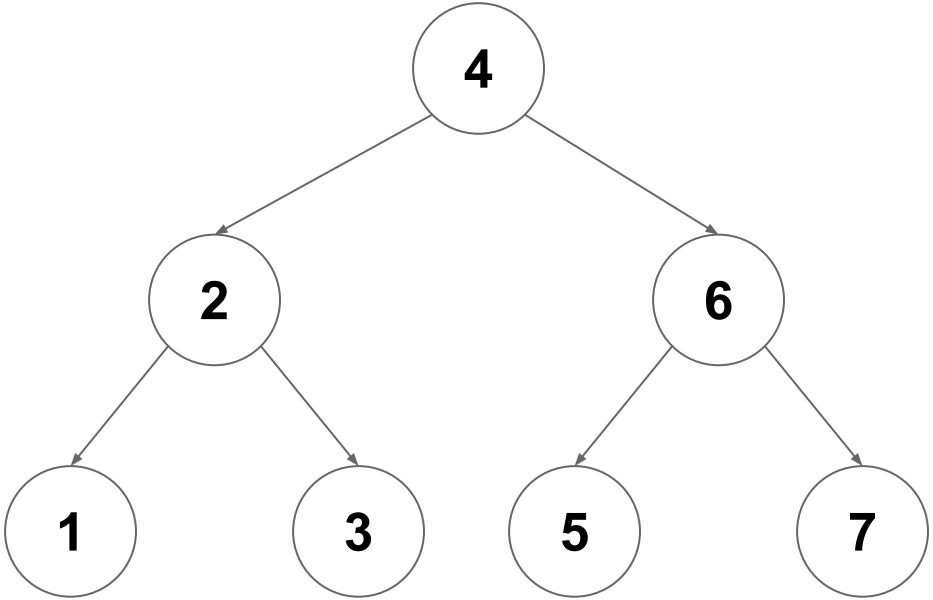
Operation	Runtime
`find`	$O(d)$
`insert`	$O(d)$
`remove`	$O(d)$

What is the runtime in terms of n? $O(n)$

$\log(n) \leq d \leq n$

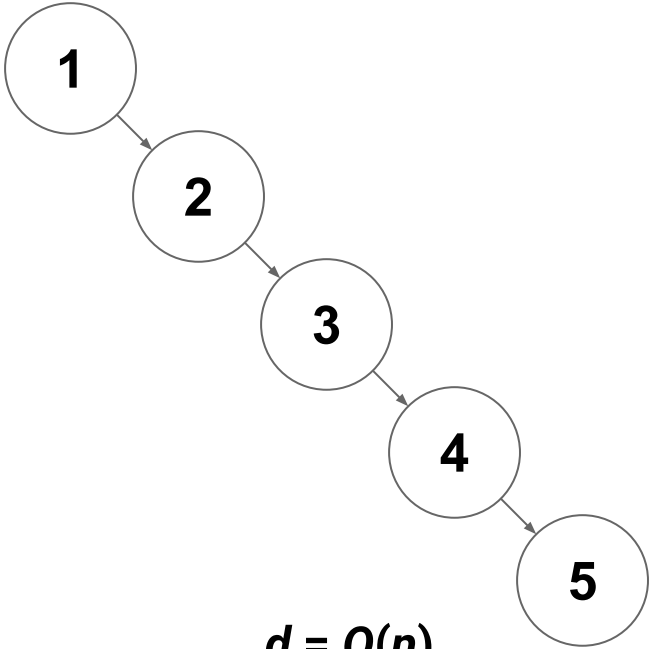
Tree Depth vs Size

If height(left) ≈ height(right)



d **=** $\boldsymbol{O(\log(n))}$

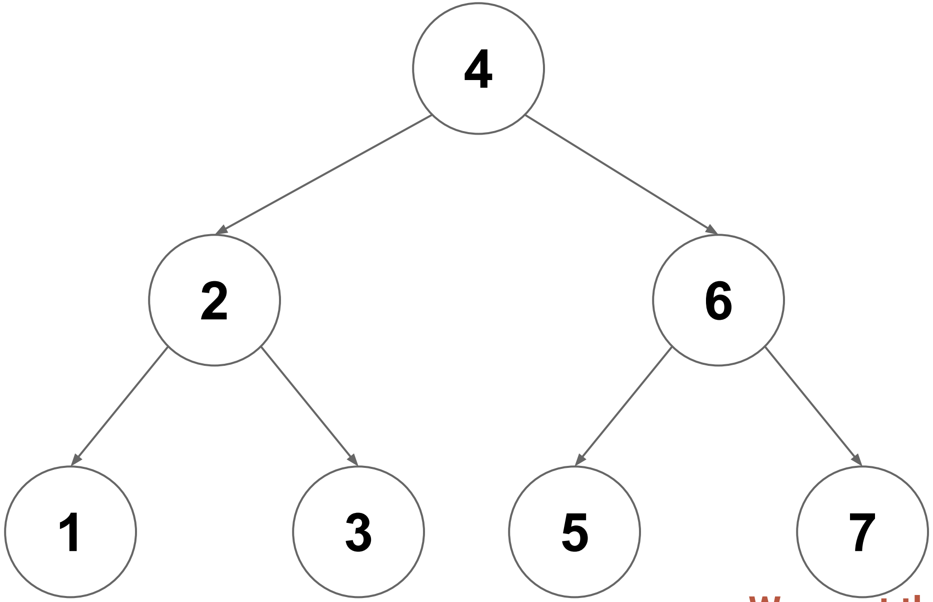
If height(left) ≪ height(right)



d **=** $\boldsymbol{O(n)}$

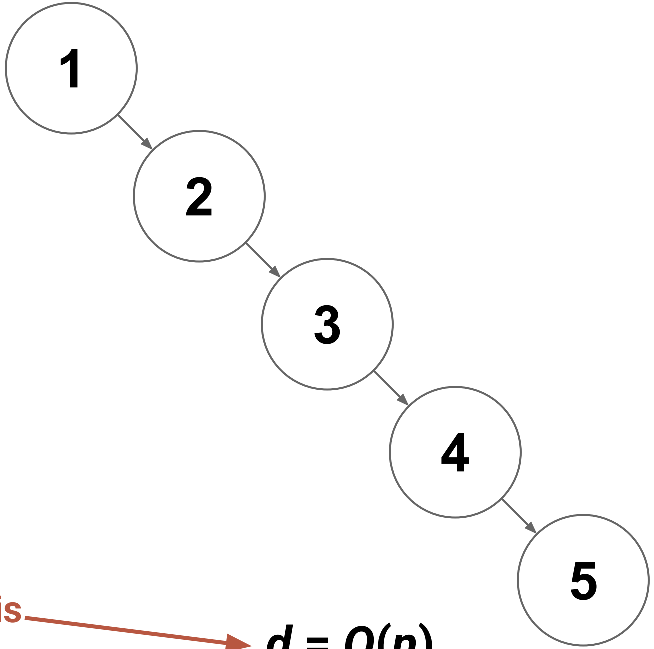
Tree Depth vs Size

If height(left) ≈ height(right)



$d = O(\log(n))$

If height(left) ≪ height(right)



$d = O(n)$

We want this, not this

Short Trees

Short Trees are good: Faster `find`, `insert`, `remove`

Short Trees

Short Trees are good: Faster **`find`**, **`insert`**, **`remove`**

How do we make our trees short?

Short Trees

Short Trees are good: Faster `find`, `insert`, `remove`

How do we make our trees short? **keep them "balanced"**

Balanced Trees

Short Trees are good: Faster **`find`**, **`insert`**, **`remove`**

How do we make our trees short? **keep them "balanced"**

What is balanced? How do we keep a tree balanced?

Balanced Trees - Two Approaches

Option 1

Keep left/right subtrees within **+/-1** of each other in height

(add a field to track amount of "imbalance")

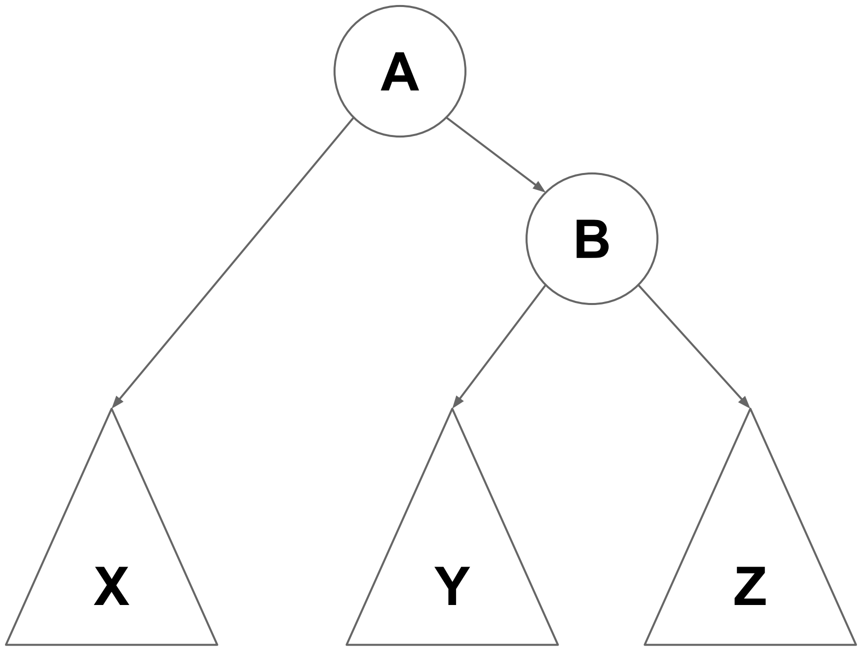
Option 2

Keep leaves at some minimum depth (***d*/2**)

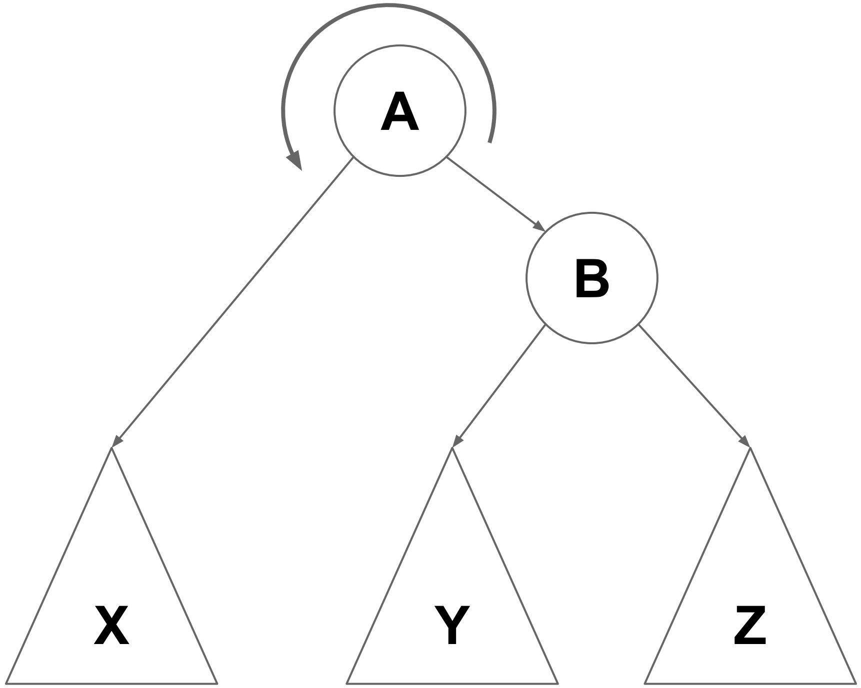
(Add a color to each node marking it as "red" or "black")

Ok…but how do we enforce this…?

Rebalancing Trees (rotations)

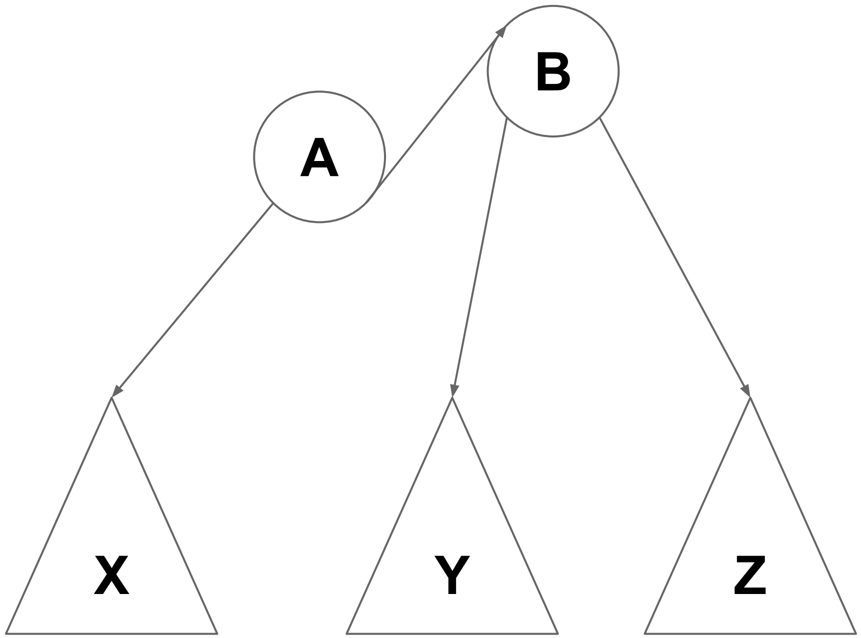


Rebalancing Trees (rotations)



Rotate(A, B)

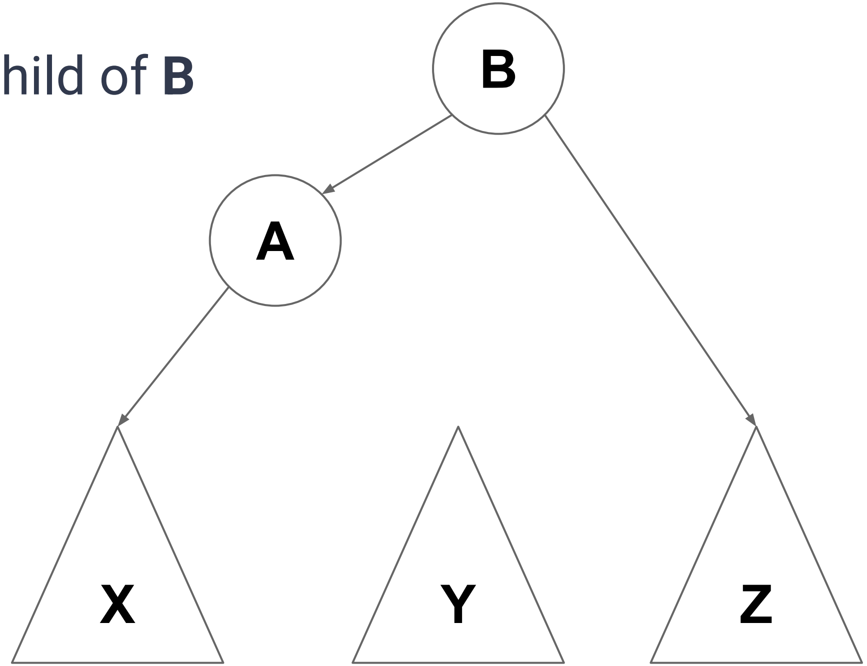
Rebalancing Trees (rotations)



Rotate(A, B)

Rebalancing Trees (rotations)

Make **A** the left child of **B**

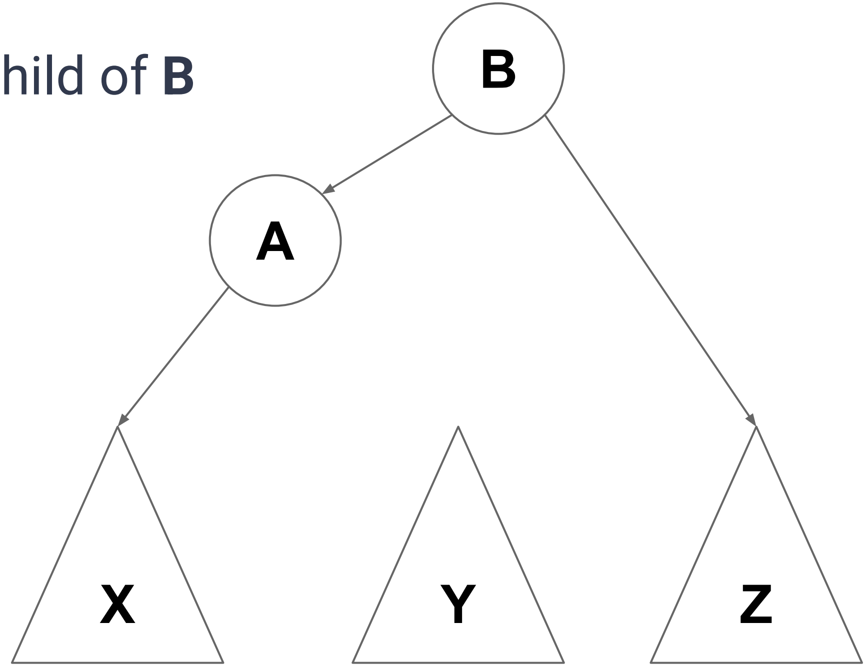


Rotate(A, B)

Rebalancing Trees (rotations)

Make **A** the left child of **B**

What about **Y**?



Rotate(A, B)

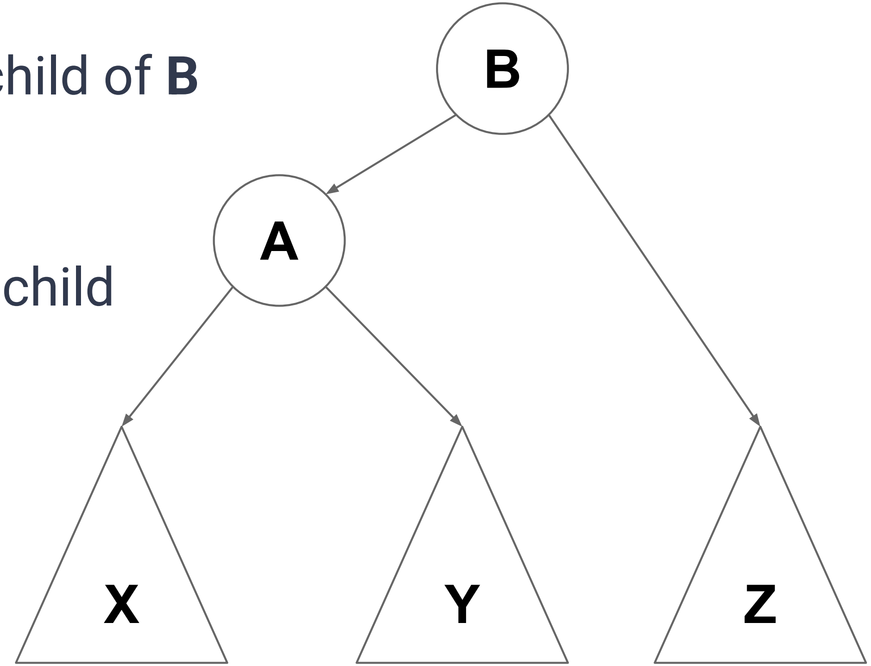
Rebalancing Trees (rotations)

Make **A** the left child of **B**

What about **Y**?

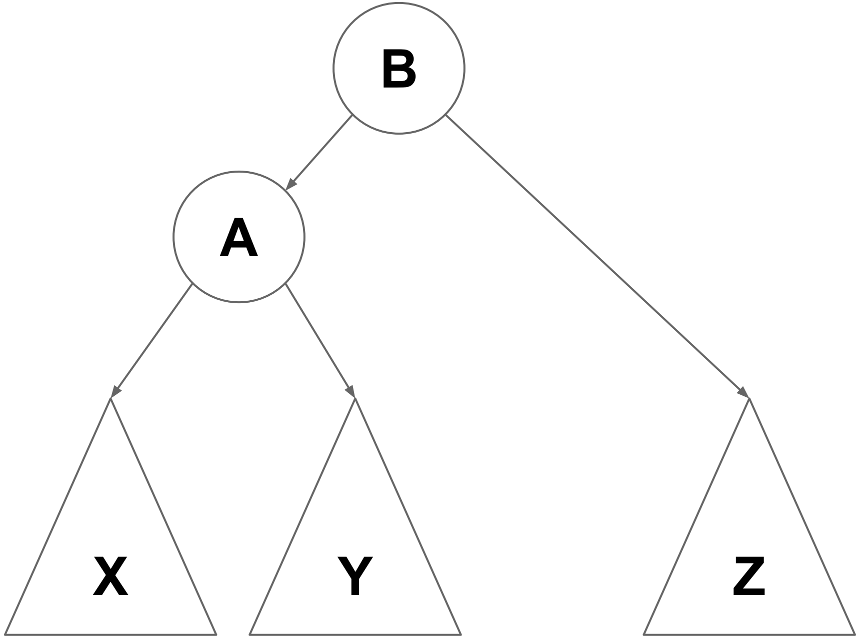
Make it the right child

of **A**



Rotate(A, B)

Rebalancing Trees (rotations)

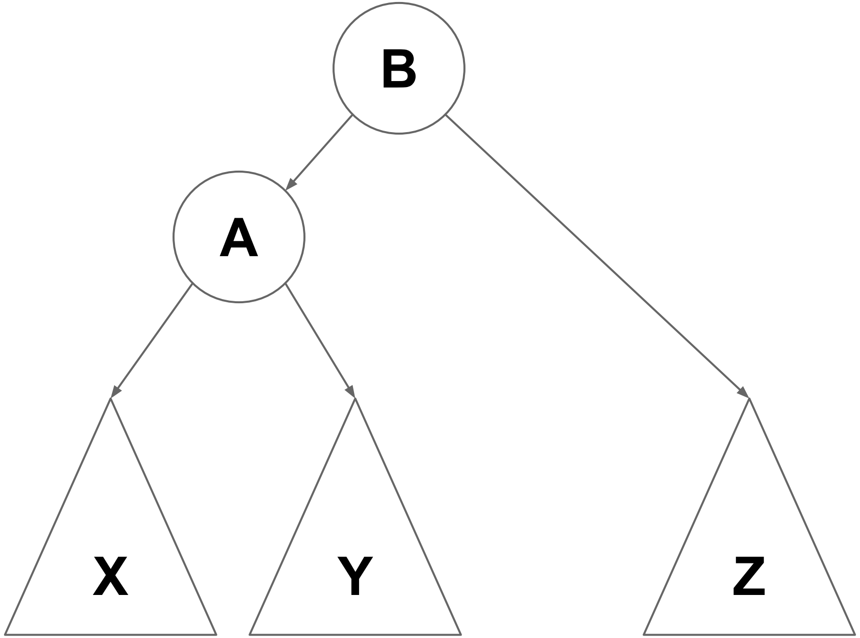


Rotate(A, B)

Rebalancing Trees (rotations)

A became **B**'s left child

B's left child became **A**'s right child



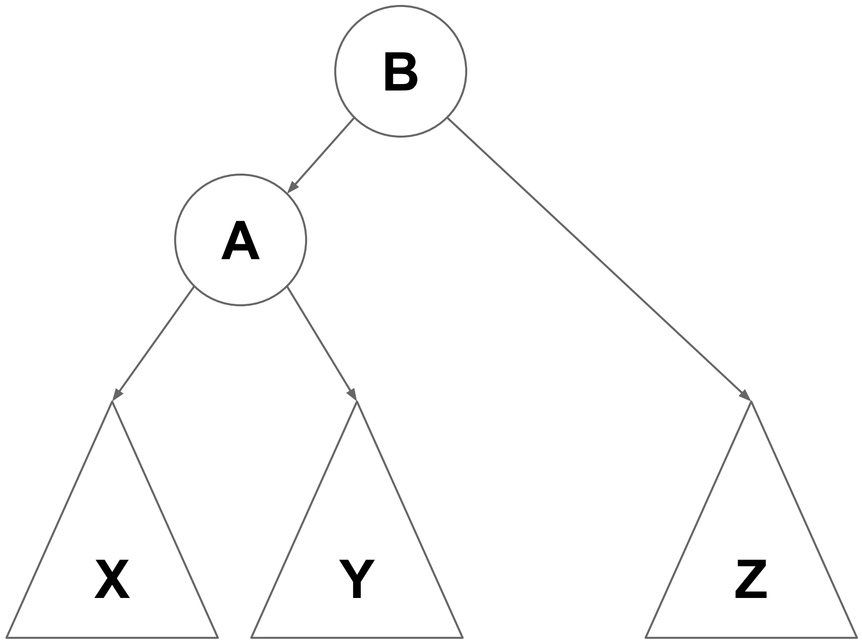
Rotate(A, B)

Rebalancing Trees (rotations)

A became **B**'s left child

B's left child became **A**'s right child

Is ordering maintained?



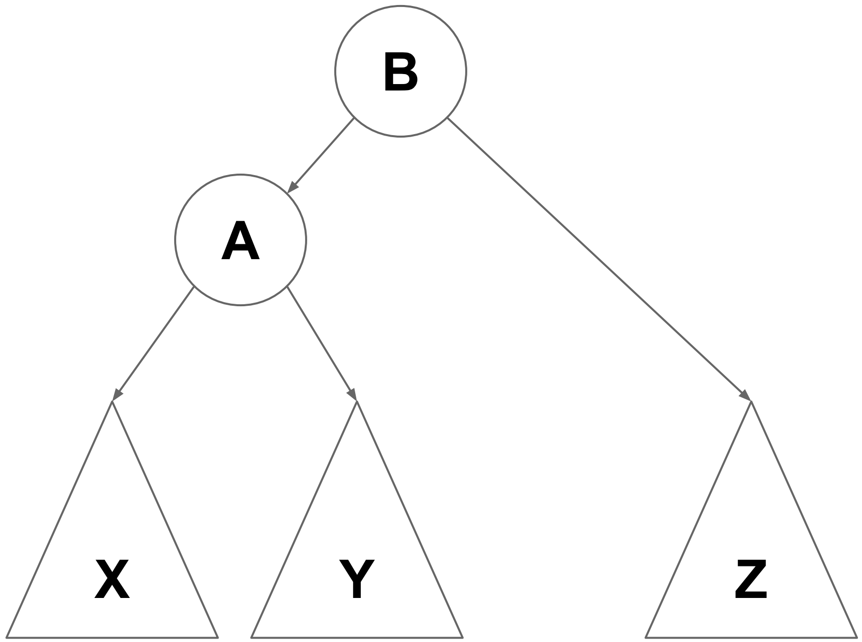
Rotate(A, B)

Rebalancing Trees (rotations)

A became **B**'s left child

B's left child became **A**'s right child

Is ordering maintained? **Yes!**



Rotate(A, B)

Rebalancing Trees (rotations)

A became **B**'s left child

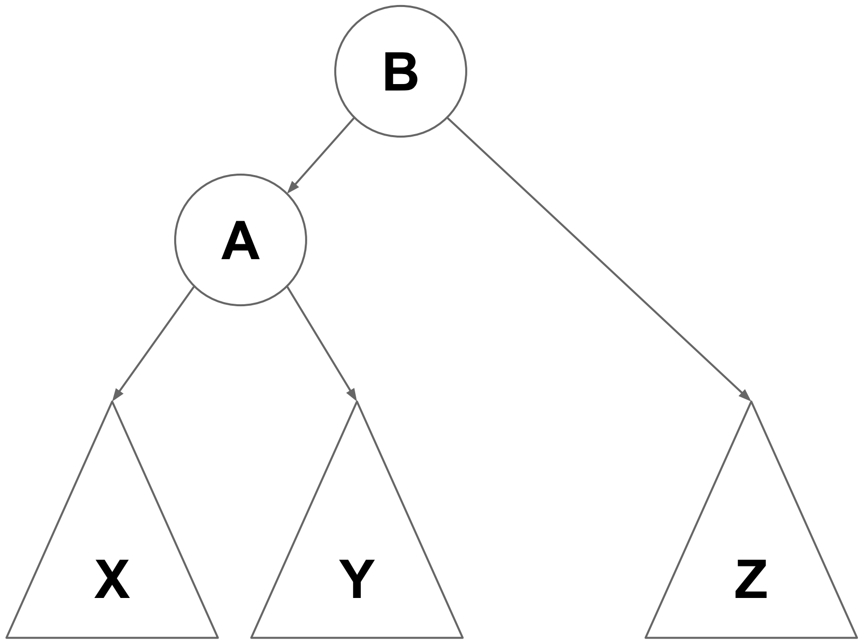
B's left child became **A**'s right child

Is ordering maintained? **Yes!**

B used to be the right child of **A**

Therefore **B** is bigger than **A**

Therefore **A** is smaller than **B** ✓



Rotate(A, B)

Rebalancing Trees (rotations)

A became **B**'s left child

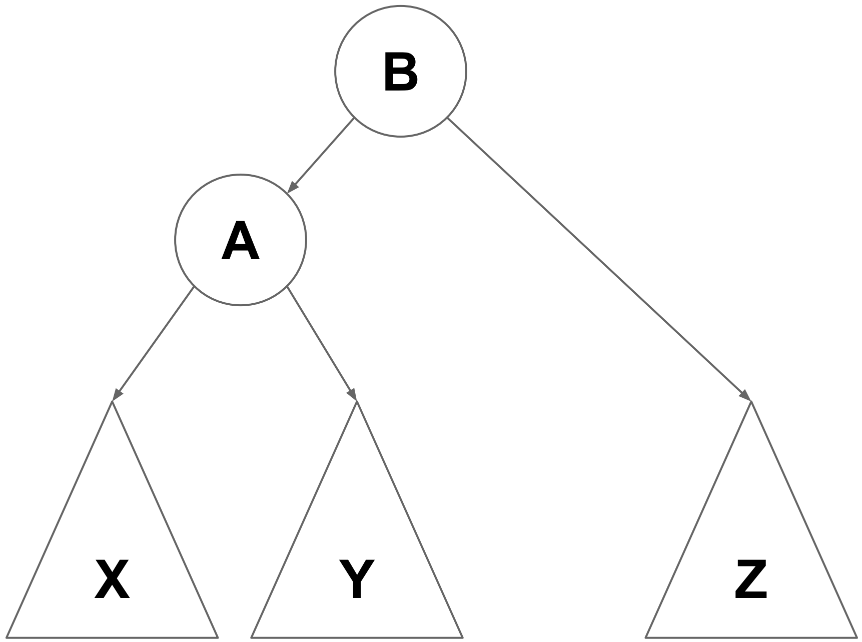
B's left child became **A**'s right child

Is ordering maintained? **Yes!**

Y used to be in the left subtree of **B**

Therefore **Y** is smaller than **B**

It is still left of **B** ✓



Rotate(A, B)

Rebalancing Trees (rotations)

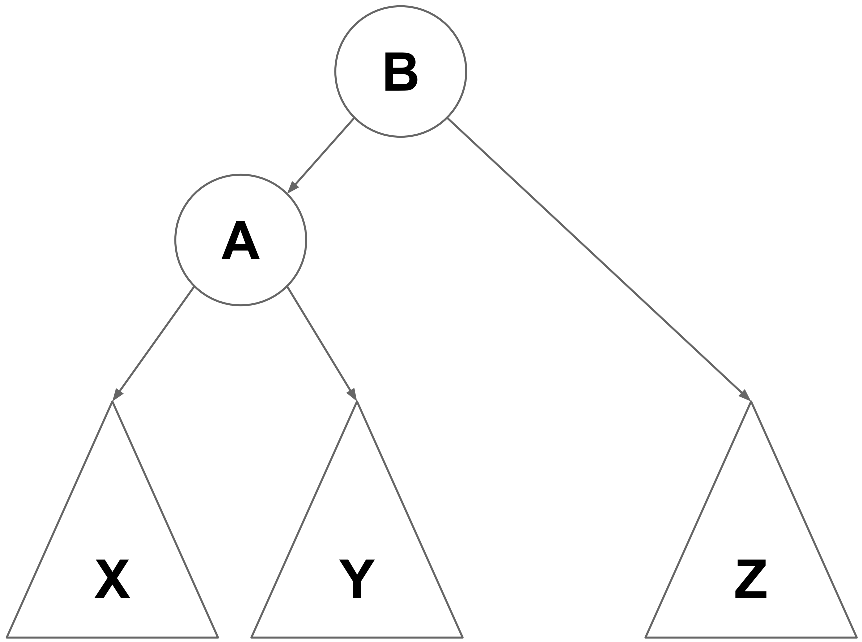
A became **B**'s left child

B's left child became **A**'s right child

Is ordering maintained? **Yes!**

Y used to be in the right subtree of **A**

It is still in the right subtree of **A** ✓



Rotate(A, B)

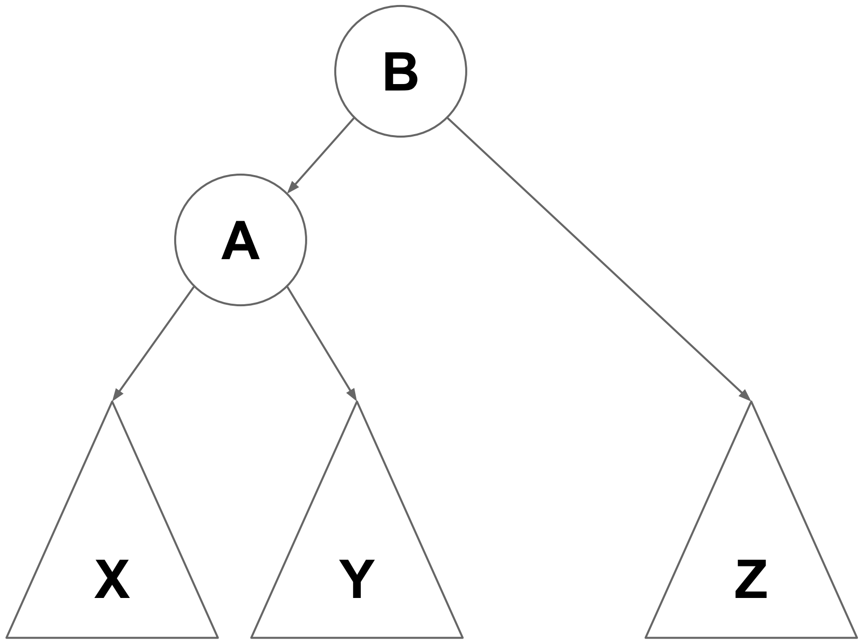
Rebalancing Trees (rotations)

A became **B**'s left child

B's left child became **A**'s right child

Is ordering maintained? **Yes!**

Complexity?



Rotate(A, B)

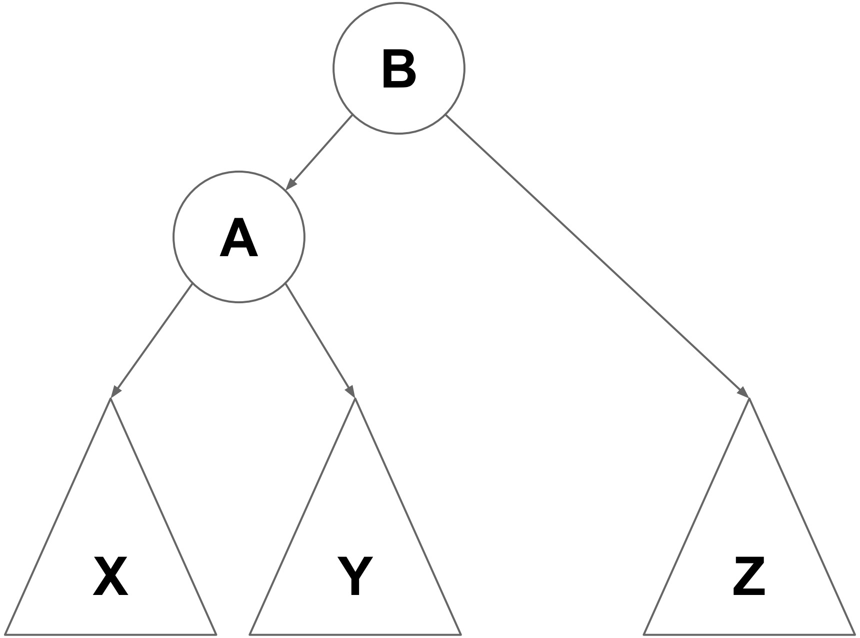
Rebalancing Trees (rotations)

A became **B**'s left child

B's left child became **A**'s right child

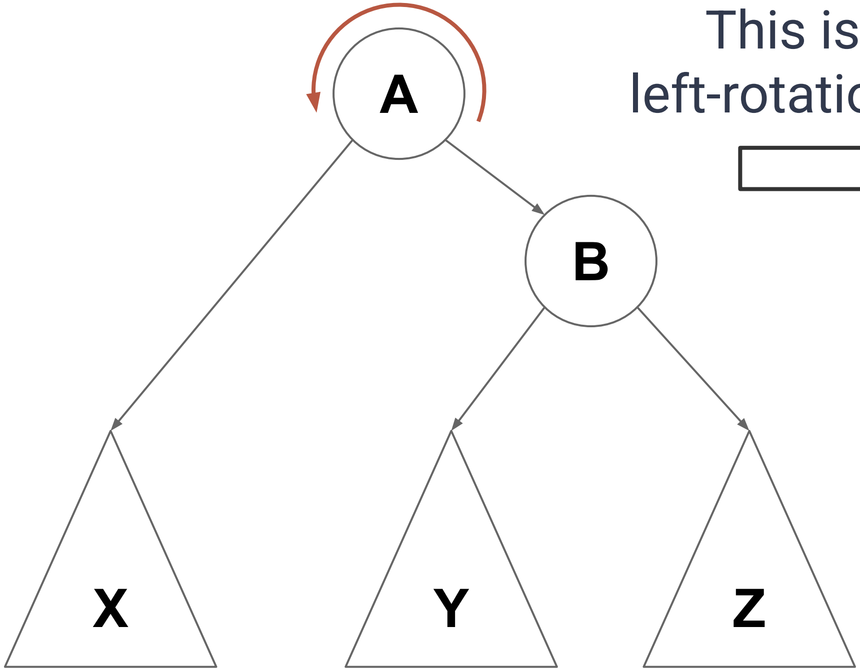
Is ordering maintained? **Yes!**

Complexity? ***O*(1)**

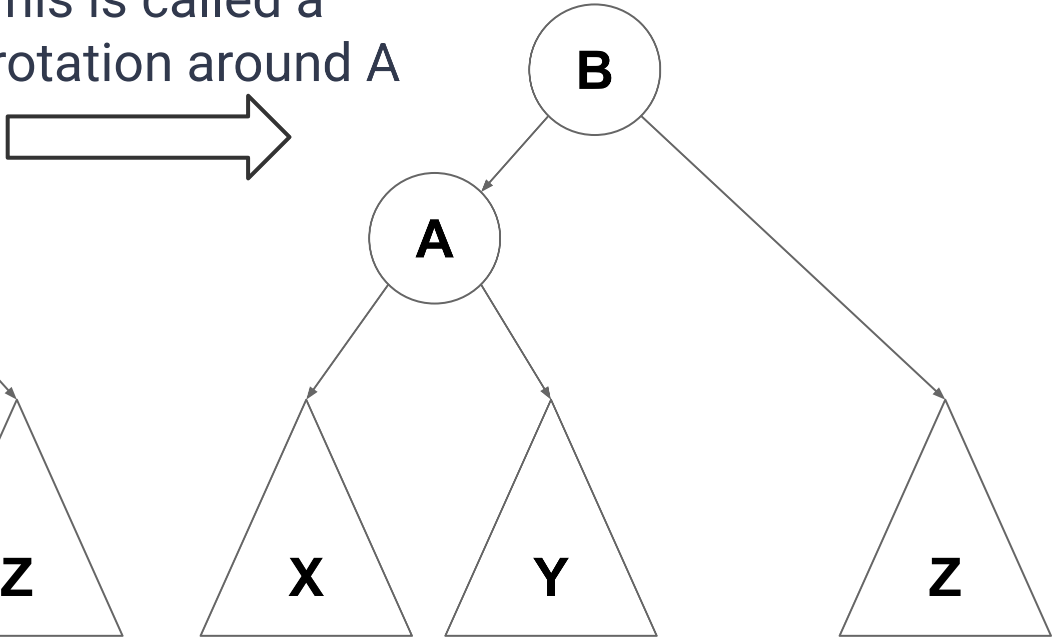


Rotate(A, B)

Rebalancing Trees (rotations)

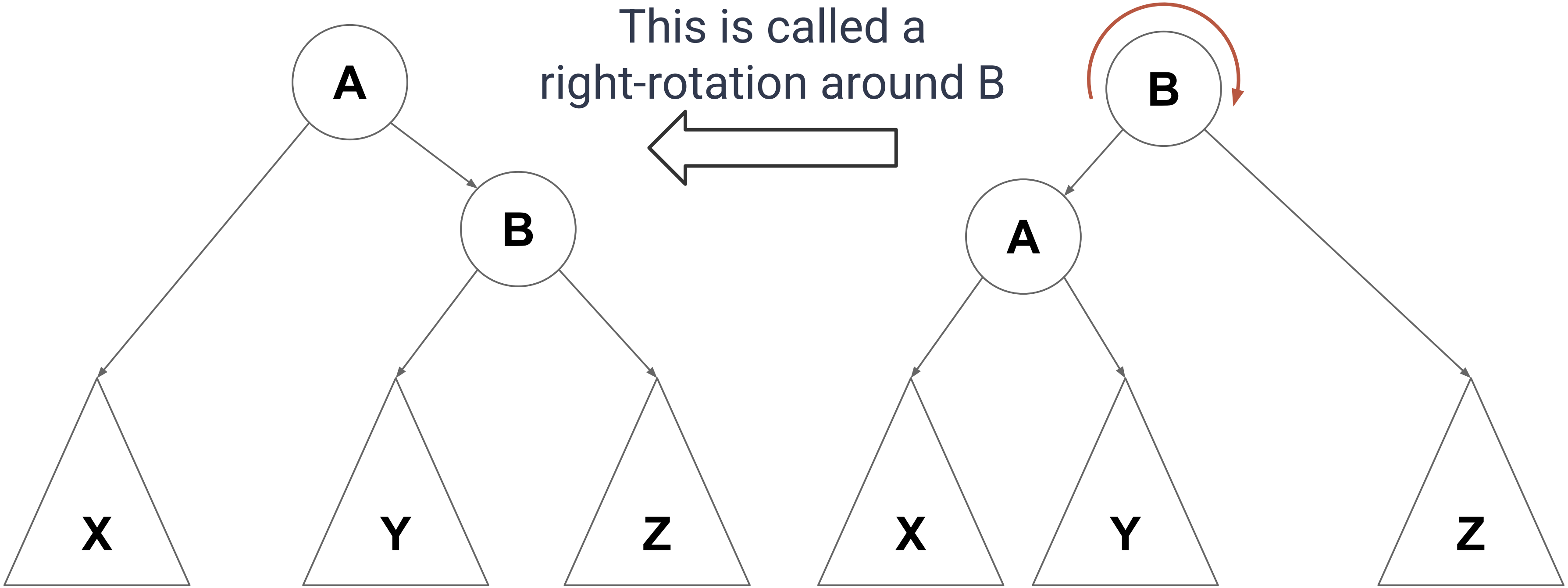


This is called a
left-rotation around A



Rebalancing Trees (rotations)

This is called a
right-rotation around B



Rebalancing Trees (rotations)

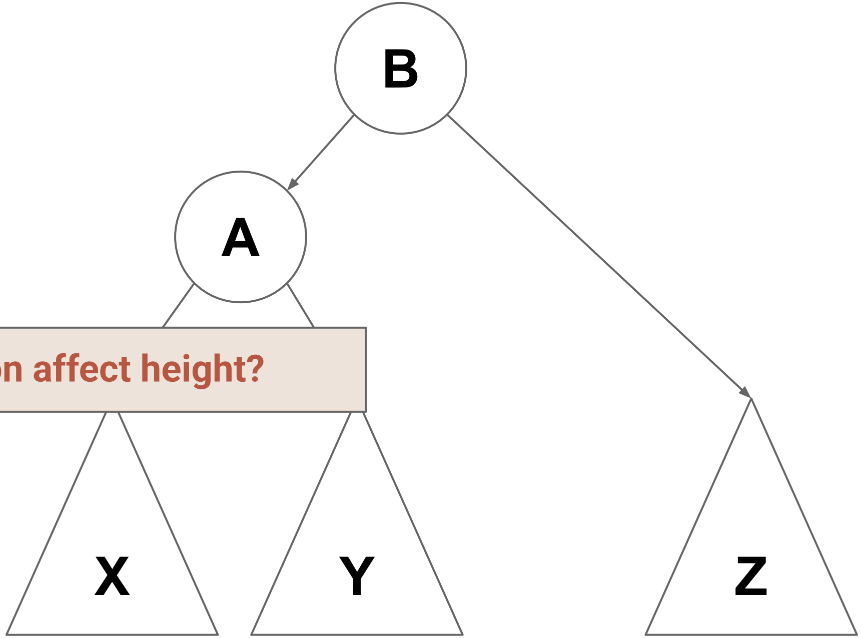
A became **B**'s left child

B's left child became **A**'s right child

Is ordering maintained? **Yes!**

Complexity? ***O*(1)**

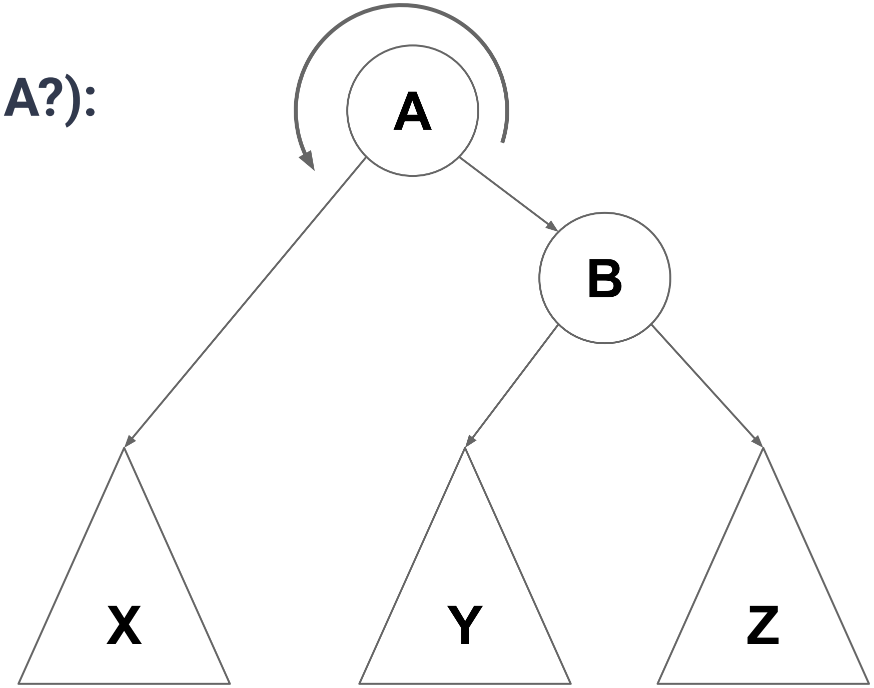
How does a rotation affect height?



Rotate(A, B)

Rebalancing Trees (rotations)

Before Rotation (what is the height of A?):

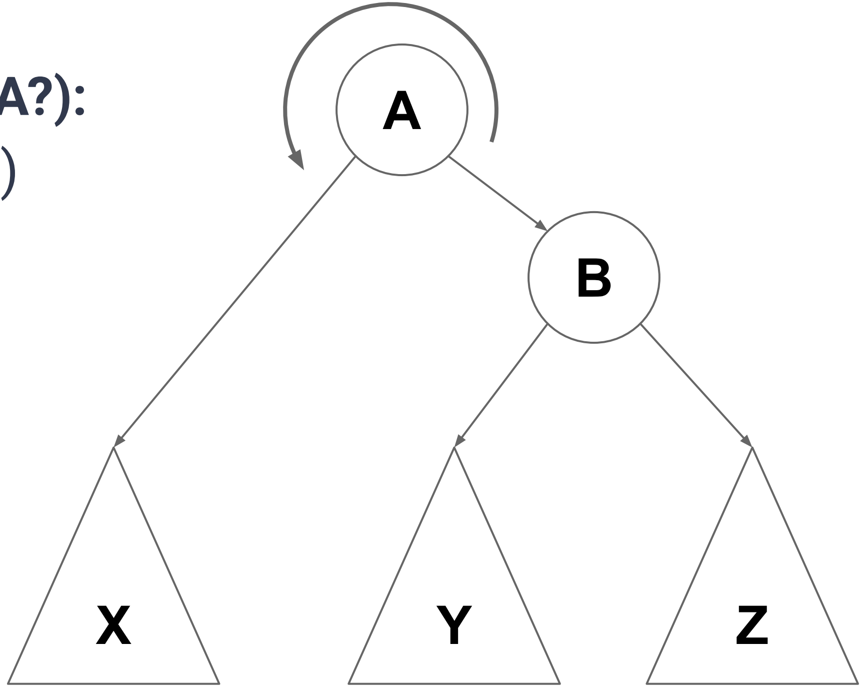


Rotate(A, B)

Rebalancing Trees (rotations)

Before Rotation (what is the height of A?):

h(A) = 1 + max(h(X), 1 + max(h(Y), h(Z))



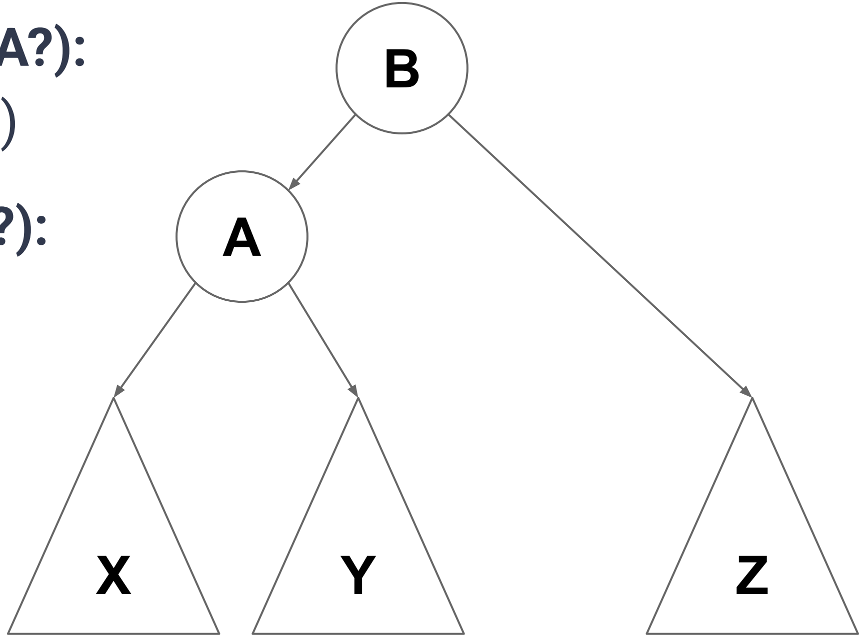
Rotate(A, B)

Rebalancing Trees (rotations)

Before Rotation (what is the height of A?):

h(A) = 1 + max(h(X), 1 + max(h(Y), h(Z))

After Rotation (what is the height of B?):



Rotate(A, B)

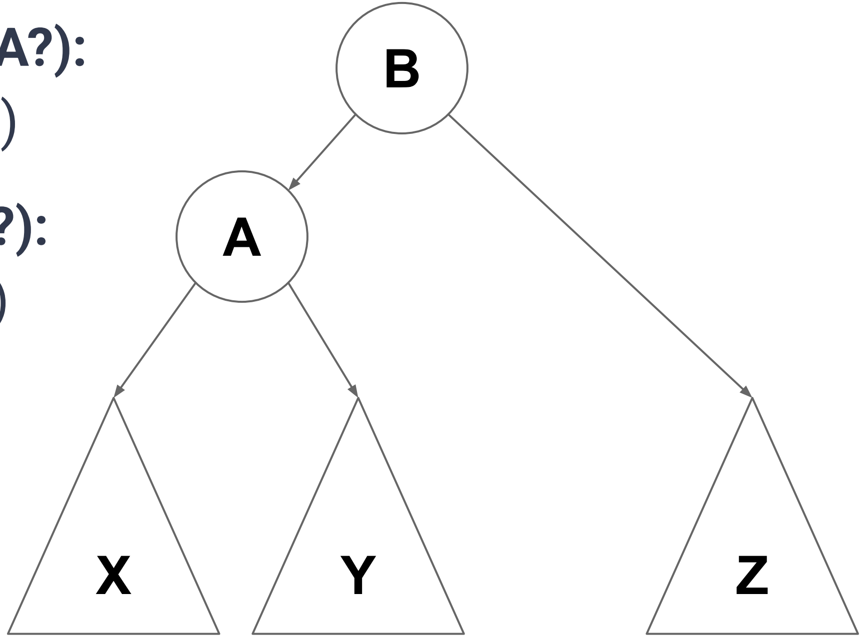
Rebalancing Trees (rotations)

Before Rotation (what is the height of A?):

h(A) = 1 + max(h(X), 1 + max(h(Y), h(Z))

After Rotation (what is the height of B?):

h(B) = 1+ max(1 + max(h(X),h(Y)), h(Z))



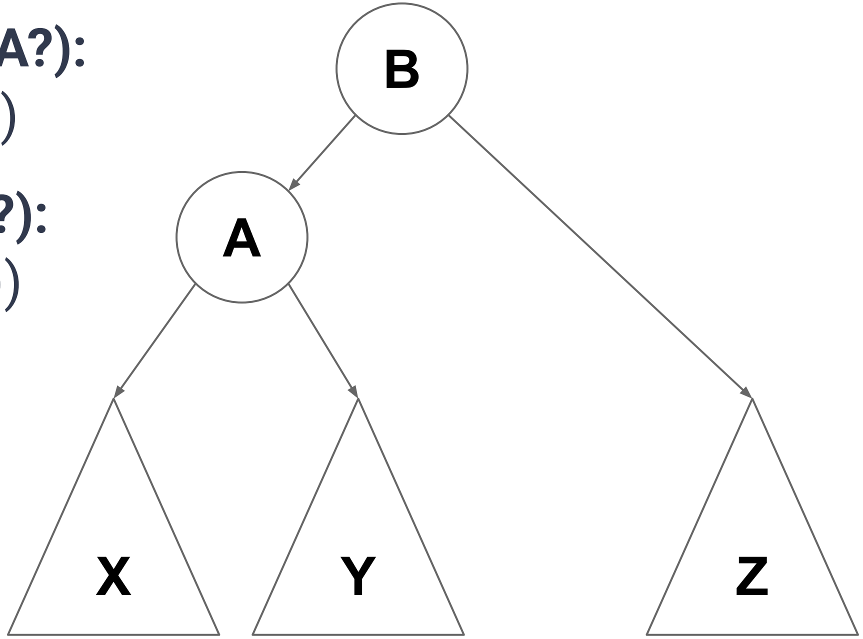
Rotate(A, B)

Rebalancing Trees (rotations)

Before Rotation (what is the height of A?):
h(A) = 1 + max(h(X), 1 + max(h(Y), h(Z))

After Rotation (what is the height of B?):
h(B) = 1 + max(1 + max(h(X),h(Y)), h(Z))

- If **X** was the tallest of **X,Y,Z** our total height increased by 1.
- If **Z** was the tallest our total height decreased by 1.
- If **X,Z** same height, or **Y** is the tallest then total is unchanged



Rotate(A, B)

Rebalancing Trees (rotations)

Before Rotation (what is the height of A?):

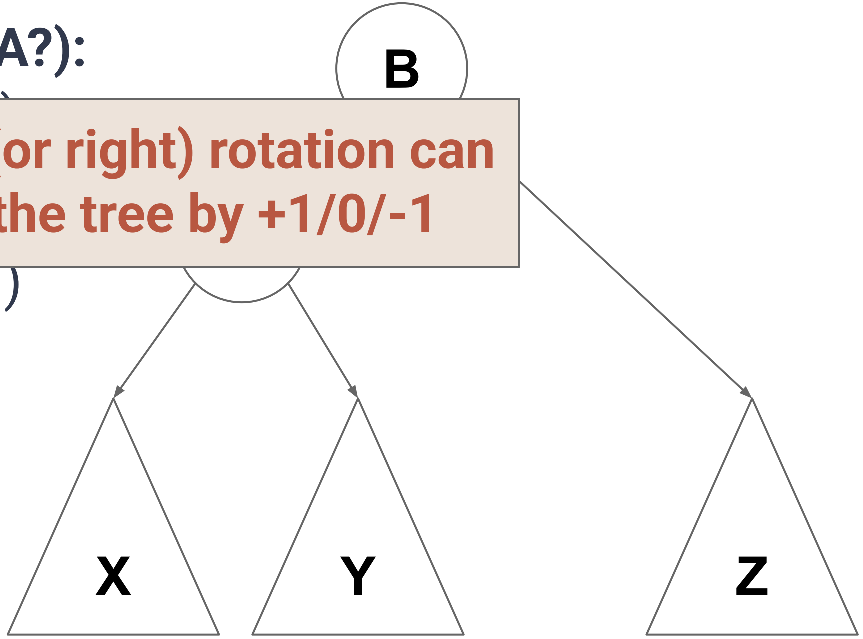
h(A) = 1 + ma

Therefore, a single left (or right) rotation can change the height of the tree by +1/0/-1

After Rotatio

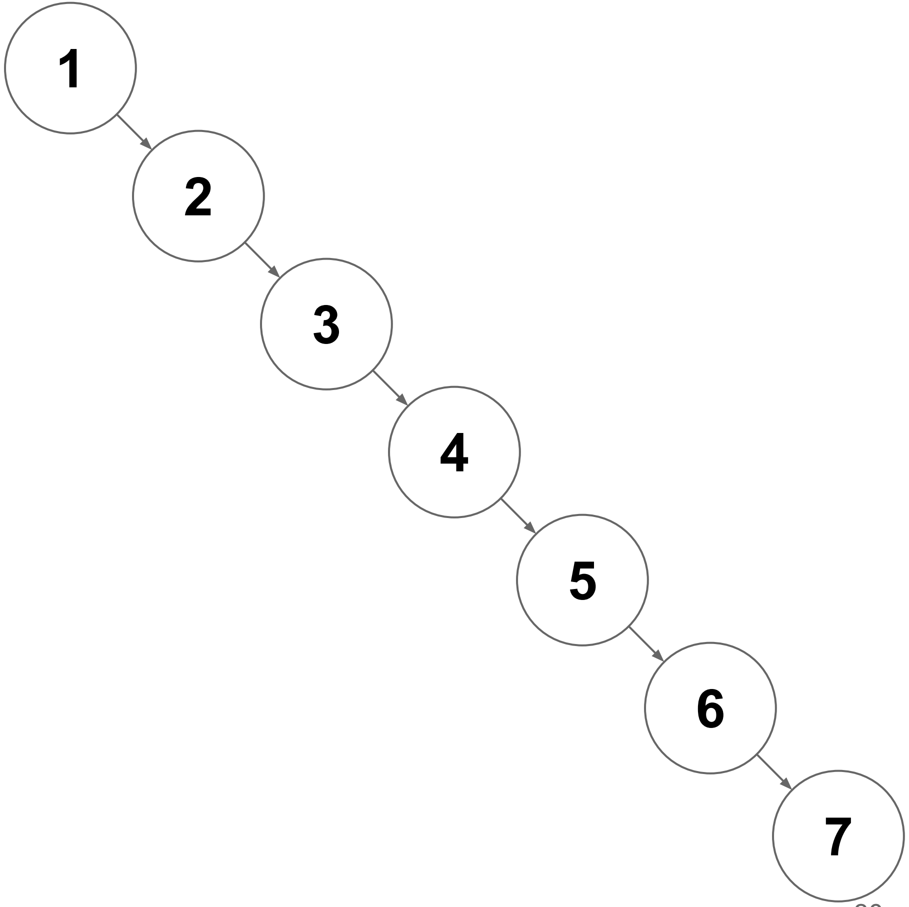
h(B) = 1 + max(1 + max(h(X),h(Y)), h(Z))

- If **X** was the tallest of **X,Y,Z** our total height increased by 1.
- If **Z** was the tallest our total height decreased by 1.
- If **X,Z** same height, or **Y** is the tallest then total is unchanged



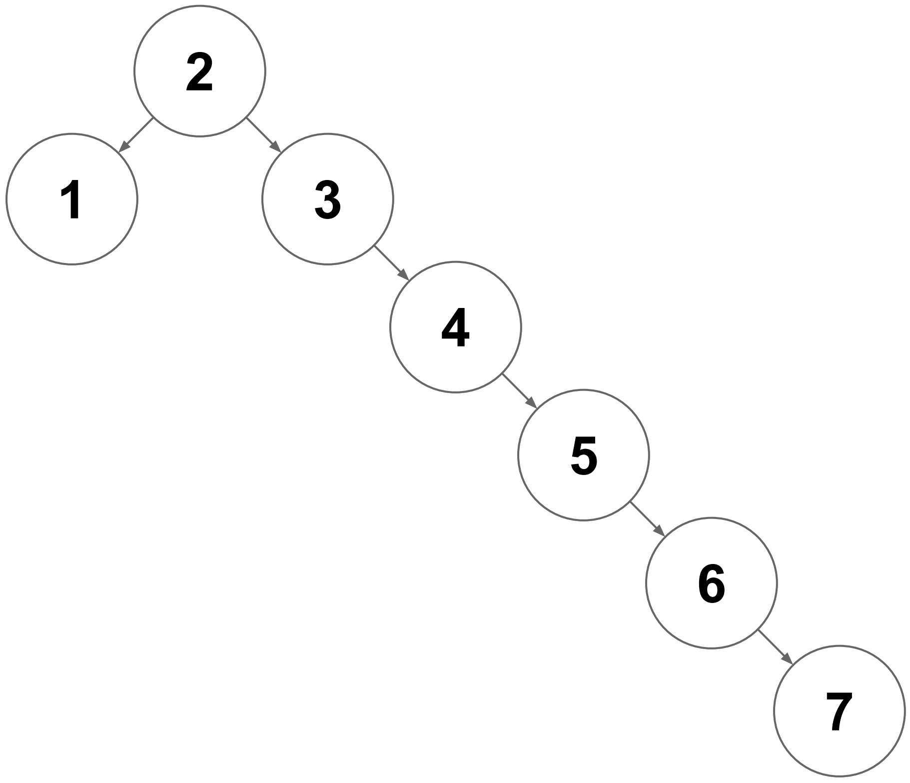
Rotate(A, B)

Rebalancing Trees



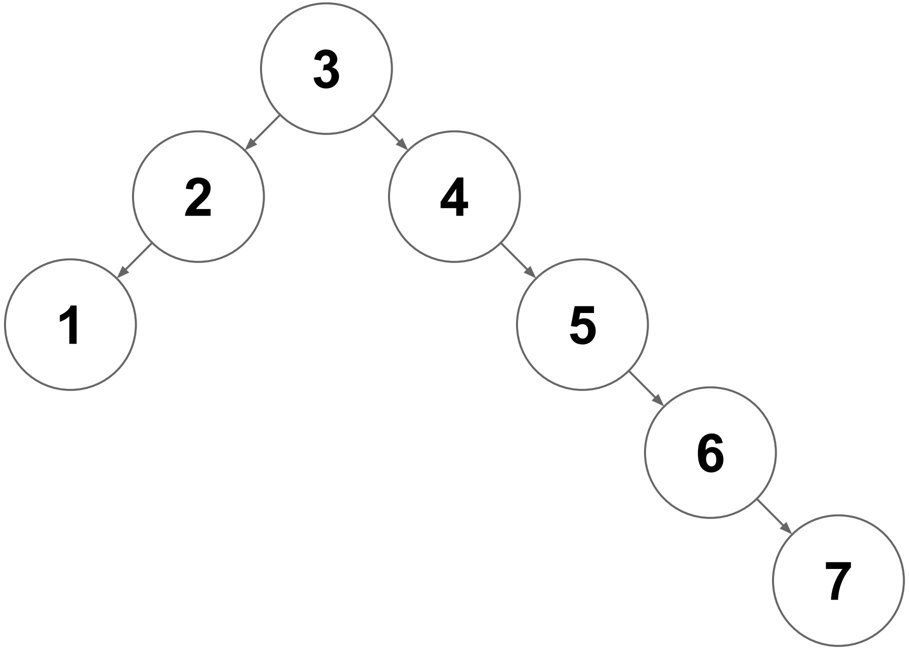
Rebalancing Trees

`Rotate(1,2)`



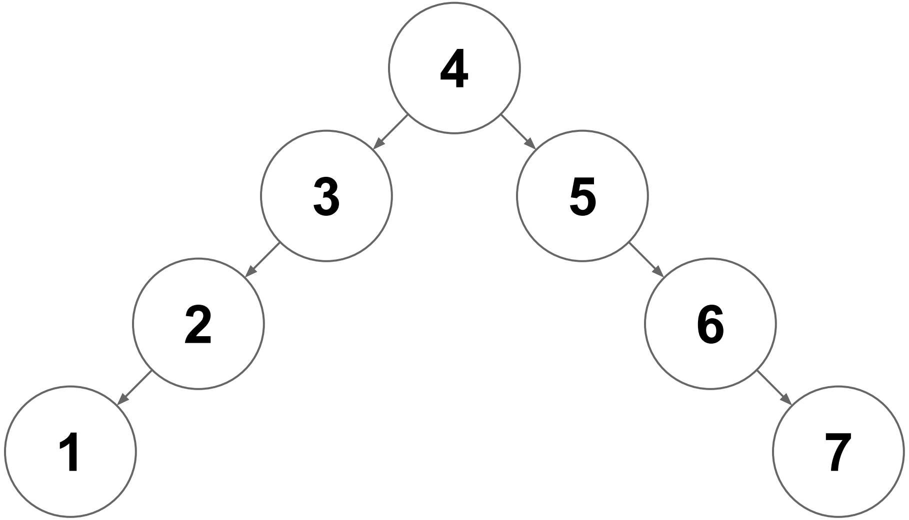
Rebalancing Trees

`Rotate(2,3)`



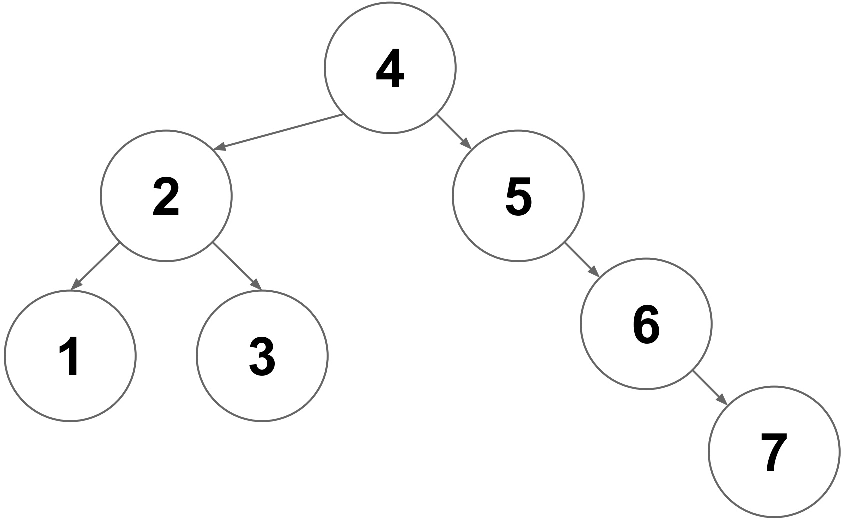
Rebalancing Trees

`Rotate(3,4)`



Rebalancing Trees

`Rotate(3,2)`



Rebalancing Trees

`Rotate(5,6)`